ENGINE

Teaching online electronics, microcontrollers and programming in Higher Education

Sprzętowa implementacja algorytmów

4. Dekoder kodu BCD na kod wyświetlacza siedmio-segmentowego. Liczniki.

Lider projektu: Politechnika Warszawska

Autor: Łukasz Mik

Akademia Nauk Stosowanych w Tarnowie

Erasmus+

GRANT NUMBER: 2020-1-PL01-KA226-HE-095653

Declaration

This laboratory instruction has been prepared in the context of the ENGINE project. Where other published and unpublished source materials have been used, these have been acknowledged.

Copyright

Funding Disclaimer

This project has been funded with support from the European Commission. This report reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

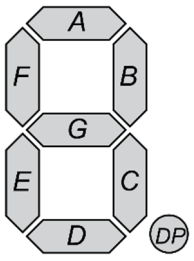
I. Kod BCD i wyświetlacz siedmio-segmentowy.

Kod BCD (ang. *Binary-Coded Decimal* - zapis dziesiętny kodowany dwójkowo) jest to sposób zapisu liczby polegający na zakodowaniu kolejnych cyfr dziesiętnych tej liczby w systemie dwójkowym. Do tego celu wykorzystuje się tylko cztery bity (półbajt). Ze względu na sposób reprezentacji liczb w systemach cyfrowych jest on powszechnie wykorzystywany w elektronice i informatyce. Taki zapis pozwala na łatwą konwersję liczby dziesiętnej na binarną i odwrotnie.

Cyfra	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Kod BCD jest najczęściej stosowany w urządzeniach elektronicznych z wyświetlaczem cyfrowym (np. w kalkulatorach, miernikach cyfrowych).

Wyświetlacz siedmio-segmentowy jest to rodzaj wyświetlacza składający się z 7 elementów (segmentów) świecących, ułożonych w kształcie umożliwiającym wyświetlanie wszystkich cyfr dziesiętnych od 0 do 9. Elementami świecącymi są najczęściej diody LED (patrz: poniższy rysunek)

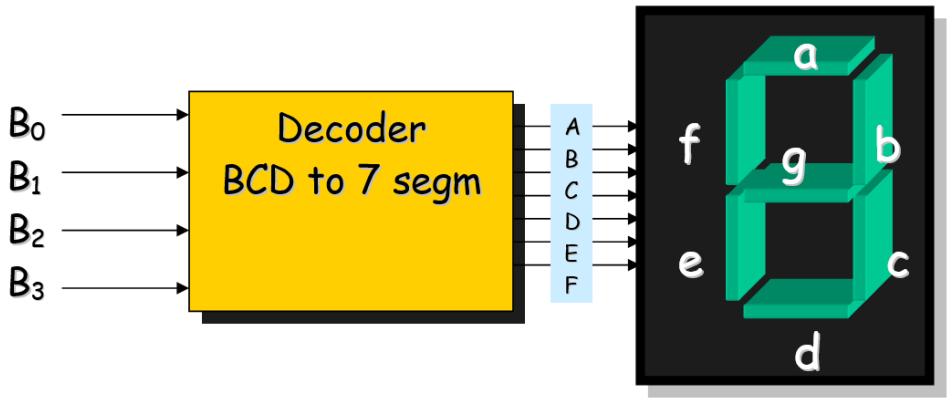


Oznaczenia segmentów pojedynczego wyświetlacza siedmio-segmentowego

Segmenty są oznaczone literami od A do G zgodnie z ruchem wskazówek zegara, rozpoczynając od najwyżej położonego. Dodatkowy 8 segment o nazwie DP służy do oznaczania przecinka w liczbie z częścią ułamkową.

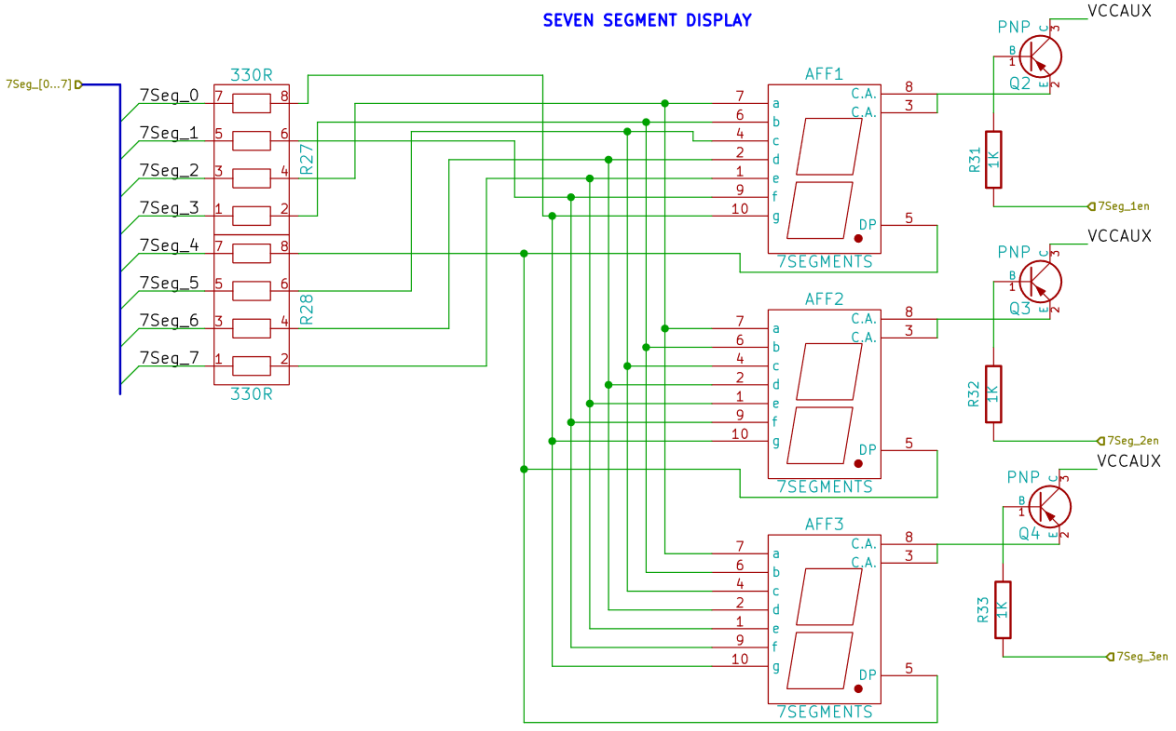
II. Dekoder kodu BCD na kod wyświetlacza siedmio-segmentowego.

Bezpośrednie podłączenie 4 linii z kodem BCD do wyświetlacza siedmio-segmentowego nie jest możliwe. Konieczne jest użycie specjalnego dekodera, który przekształci ciąg 4-bitowy podawany na jego wejściu na ciąg 7-bitowy generowany na jego wyjściu. Uproszczony schemat połączenia dekodera z wyświetlaczem pokazano na poniższym rysunku.



Uproszczony schemat połączeń dekodera i wyświetlacza

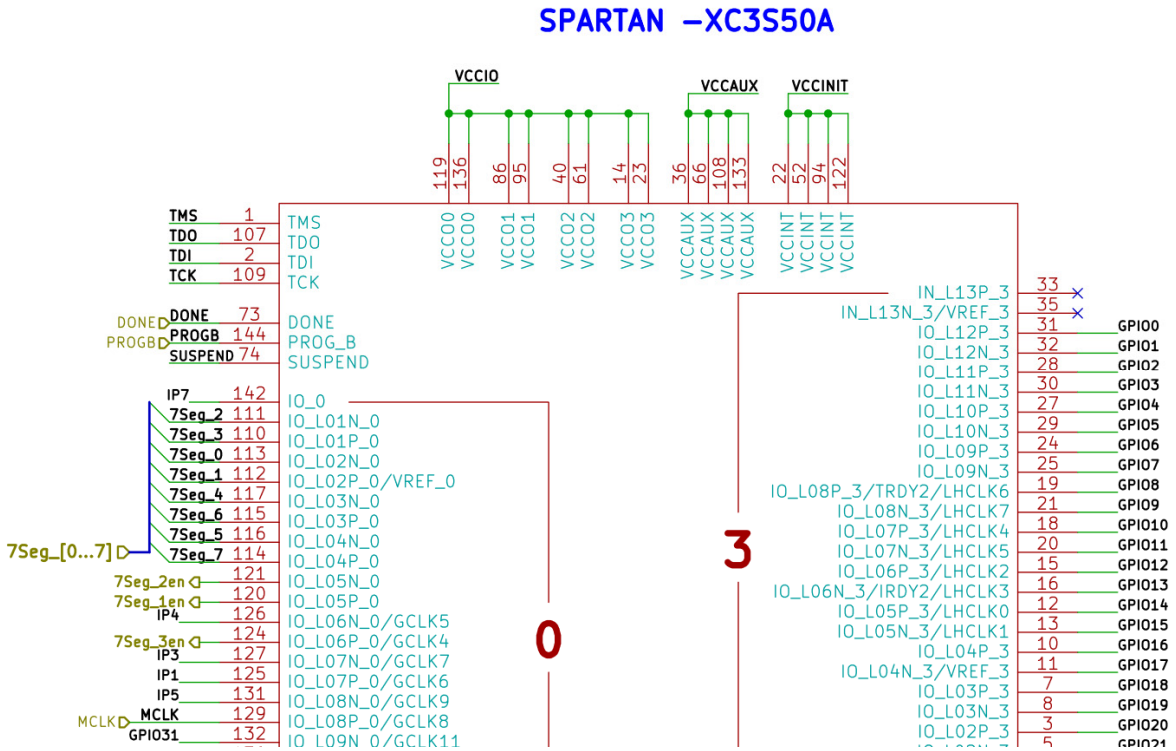
Na płycie ewaluacyjnej Numato Elbert v2 są umieszczone 3 wyświetlacze 7-segmentowe, które współdzielą te same linie portów układu FPGA.



Fragment schematu przedstawiający sekcję 3 wyświetlaczy 7-segmentowych

Jak można zauważyć na powyższym rysunku wszystkie wyświetlacze są typu wspólna anoda. W związku tym do zaświecenia każdego z segmentów od **a** do **g** konieczne jest zwarcie tych wyprowadzeń do masy (wystawienie na pinie układu FPGA stanu 0 logicznego). Bazy tranzystorów Q2, Q3 i Q4 są połączone do pinów układu FPGA i oznaczone jako linie **en** (enable). Wybór aktywnego wyświetlacza odbywa się przez podanie 0 logicznego na bazę jednego z tranzystorów. Drabinka rezystorów R27, R28 jest użyta do ograniczenia prądu płynącego przez diody LED w wyświetlaczu siedmiosegmentowym jak również portu wyjściowego układu FPGA.

Od strony układu FPGA linie wyświetlacza zostały połączone do pinów od 110 do 117. Linie *enable* zostały połączone do pinów 120, 121 i 124. Producent na stronie produktu dostarcza odpowiedniego pliku z definicjami nazw portów wraz z przypisanymi numerami pinów.



Fragment schematu przedstawiający podłączenie linii wyświetlacza 7-segmentowego do układu FPGA

III. Implementacja dekodera BCDto7SEG w języku VHDL.

Do obsługi wyświetlaczy podłączonych do układu Spartan3A na płytce Elbert potrzebne jest zdefiniowanie następujących portów:

- 4 – bitowy port wejściowy o nazwie *BCD*, który fizycznie podłączymy do przełączników typu DIP.
- 8 – bitowy port wyjściowy o nazwie *SSEG*, którego podłączymy do odpowiednich segmentów wyświetlacza (uwzględniamy kropkę DP jako 8 bit)
- 3 – bitowy port wyjściowy o nazwie *EN,* odpowiedzialny za wybór aktywnego wyświetlacza.

Część nagłówkowa (deklaracja użytych bibliotek) wraz deklaracją jednostki projektowej będzie wyglądać następująco:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity bcdto7seg is
port
(
	BCD: in std_logic_vector(3 downto 0);
	SSEG: out std_logic_vector(7 downto 0);
	EN : out std_logic_vector(1 downto 0);
);
end bcdto7seg;
```

Opis architektury powinien uwzględniać wszystkie stany na wejściu BCD i przypisywać odpowiednie stany na wyjściu SSEG. Jeśli chcemy wyświetlać cyfry na jednym z wyświetlaczy to wówczas wystarczy wyzerować jeden bit wektora EN. Szablon opisu architektury został zaprezentowany niżej. Należy pamiętać, że wyświetlacz jest typu wspólna anoda, dlatego stanem aktywnym na wejściu każdego segmentu jest 0 logiczne. Przełącznik typu DIP, który jest podłączony do portu BCD naszego dekodera również pracuje w logice odwrotnej, dlatego w architekturze wykorzystamy pomocniczy sygnał *BCD_temp*, za pomocą którego zanegujemy bity z wejścia BCD.

```
architecture Behavioral of bcdto7seg is
signal BCD_temp : std_logic_vector(3 downto 0);
begin
EN <= "110";
BCD_temp <= not BCD;
process(BCD_temp)
begin
	case BCD_temp is
		--------------------edcpbafg
		when "0000"=>SSEG<="00000011";
		when "0001"=>SSEG<="10011111";
		when "0010"=>SSEG<="00100101";
		when "0011"=>SSEG<="00001101";
		when "0100"=>SSEG<="10011001";
		when "0101"=>SSEG<="01001001";
		when "0110"=>SSEG<="01000001";
		when "0111"=>SSEG<="00011111";
		when "1000"=>SSEG<="00000001";
		when "1001"=>SSEG<="00001001";
		when others=>SSEG<="11111111";
end case;
end process;
end Behavioral;
```

Do stworzenia projektu dekodera konieczne jest jeszcze wygenerowanie pliku UCF, który będzie zawierał połączenia portów jednostki projektowej *bcdto7seg* do pinów fizycznych układu. Należy pamiętać, że każdy bit wektora wejściowego lub wyjściowego musi być przypisany oddzielnie. Zawartość pliku UCF została przedstawiona poniżej.

```
NET "BCD[0]"  LOC = P70   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "BCD[1]"  LOC = P69   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "BCD[2]"  LOC = P68   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "BCD[3]"  LOC = P64   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;

NET "SSEG[7]"  LOC = P117  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SSEG[6]"  LOC = P116  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SSEG[5]"  LOC = P115  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SSEG[4]"  LOC = P113  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SSEG[3]"  LOC = P112  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SSEG[2]"  LOC = P111  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SSEG[1]"  LOC = P110  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SSEG[0]"  LOC = P114  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;

NET "EN[2]"  LOC = P124 | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = FAST ;
NET "EN[1]"  LOC = P121 | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = FAST ;
NET "EN[0]"  LOC = P120  | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = FAST ;
```

W środowisku ISE WebPack należy wykonać następujące kroki:

- utworzyć projekt,
- wybrać odpowiedni model układu docelowego,
- utworzyć nowy plik źródłowy (VHDL module)
- wkleić deklarację jednostki projektowej wraz z opisem jej architektury

- utworzyć nowy plik z przypisaniem sygnałów do pinów (*Implementation Constraints File*)
- Skompilować projekt i wygenerować plik konfiguracyjny *.bit lub *.bin
- Zaprogramować układ docelowy przy użyciu programu ElbertV2Config.

Po wykonaniu tych czynności należy sprawdzić czy projekt działa prawidłowo. W przypadku problemów z kompilacją należy posłużyć się plikami źródłowymi, dołączonymi do ćwiczenia.

ZADANIE 1: Bazując na dotychczas zdobytej podczas tych i poprzednich zajęć wiedzy spróbuj dodać do projektu obsługę drugiego wyświetlacza i kolejnej sekcji 4 przełączników typu DIP. Pamiętaj, że wyświetlanie na 2 wyświetlaczach jednocześnie będzie wymagało multipleksowania, czyli szybkiej zmiany bitów w wektorze EN, odpowiedzialnego za wybór aktywnego wyświetlacza. Częstotliwość przełączania musi być na tyle duża, aby oko ludzkie nie było w stanie zauważyć migotania diod.

IV. Implementacja licznika binarnego w języku VHDL.

Do implementacji liczników konieczne jest wykorzystanie dodatkowych bibliotek, umożliwiających wykonywanie operacji arytmetycznych na wektorach interpretowanych jako liczby bez znaku.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

W deklaracji jednostki projektowej licznika zamiast wejścia BCD dajemy wejście z zegara CLK.

```
entity cntmod10 is
port
(
	CLK : in std_logic_vector(3 downto 0);
	SSEG : out std_logic_vector(7 downto 0);
	EN : out std_logic_vector(2 downto 0)
);
end cntmod10;
```

W kodzie opisu architektury definiujemy 2 sygnały: *clk_1Hz*, który będzie odpowiedzialny za generowanie sygnału zegarowego 1 Hz oraz *counter*, który będzie zliczał od 0 do 9 (licznik modulo 10).

```
architecture Behavioral of cntmod10 is
	signal clk_1Hz : std_logic:='0';
	signal counter : std_logic_vector(3 downto 0):="0000";
begin

...

end Behavioral;
```

W ciele architektury umieszczamy 3 procesy. Zadaniem pierwszego będzie wygenerowanie sygnału zegarowego o częstotliwości 1 Hz.

```
process(clk)
 variable clock_cnt : integer:=0;
 begin
    if rising_edge(clk) then
        if clock_cnt < 6000001 then
            clock_cnt := clock_cnt+1;
        else
            clock_cnt := 0;
            clk_1Hz <= not(clk_1Hz);
        end if;
    end if;
end process;
```

Drugi proces będzie odpowiedzialny za zliczanie cykli zegara 1 Hz.

```
process(clk_1Hz)
begin
    if rising_edge(clk_1Hz) then
        if counter < 10 then
            counter <= counter + 1;
        else
            counter <= "0000";
        end if;
    end if;
end process;
```

Wartość licznika *counter* jest inkrementowana z każdym zboczem narastającym sygnału clk_1Hz. Po osiągnięciu wartości 10 jest natychmiast zerowany dlatego nazywamy go licznikiem modulo 10 – zlicza w zakresie od 0 do 9.

Zadaniem trzeciego procesu będzie wyświetlanie stanu licznika z procesu drugiego na wyświetlaczu siedmio-segmentowym.

```
process(counter)
begin
      case counter is
            --------------------abcdefgp
            when "0000"=>SSEG<="00000011";
            when "0001"=>SSEG<="10011111";
            when "0010"=>SSEG<="00100101";
            when "0011"=>SSEG<="00001101";
            when "0100"=>SSEG<="10011001";
            when "0101"=>SSEG<="01001001";
            when "0110"=>SSEG<="01000001";
            when "0111"=>SSEG<="00011111";
            when "1000"=>SSEG<="00000001";
            when "1001"=>SSEG<="00001001";
            when others=>SSEG<="11111111";
      end case;
end process;
```

Zmodyfikowany plik UCF będzie miał następującą postać:

```
NET "CLK"  LOC = P129  | IOSTANDARD = LVCMOS33 | PERIOD = 12MHz;

NET "SSEG[7]"  LOC = P117  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SSEG[6]"  LOC = P116  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SSEG[5]"  LOC = P115  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SSEG[4]"  LOC = P113  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SSEG[3]"  LOC = P112  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SSEG[2]"  LOC = P111  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SSEG[1]"  LOC = P110  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SSEG[0]"  LOC = P114  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;

NET "EN[2]"  LOC = P124 | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = FAST ;
NET "EN[1]"  LOC = P121 | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = FAST ;
NET "EN[0]"  LOC = P120  | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = FAST ;
```

Podobnie jak w punkcie III utwórz projekt dodając odpowiednie pliki. Po uzupełnieniu kodów źródłowych skompiluj projekt i wygeneruj plik konfiguracyjny. W przypadku, gdyby podczas kompilacji wystąpiły błędy skorzystaj z plików źródłowych udostępnionych do zajęć. Sprawdź czy układ działa prawidłowo tj. licznik jest inkrementowany co sekundę, a jego wartość wyświetla się na wyświetlaczu 7-segmentowym.

ZADANIE 2: Zmodyfikuj projekt z punktu IV tak, aby wewnątrz architektury były 2 liczniki modulo 10. Jedne zliczający w górę i jeden zliczając w dół. Inkrementacja i dekrementacja obu liczników ma się odbywać z częstotliwością 1 Hz. Pamiętaj, że wartości liczników muszą być wyświetlane na oddzielnych wyświetlaczach.

References

- P. Zbysiński, J. Pasierbiński – *Układy programowalne – pierwsze kroki.* Wydawnictwo BTC, Warszawa 2004

- Elbert V2 S3A FPGA Development Board*, https://numato.com/product/elbert-v2-spartan-3a-fpga-development-board/*